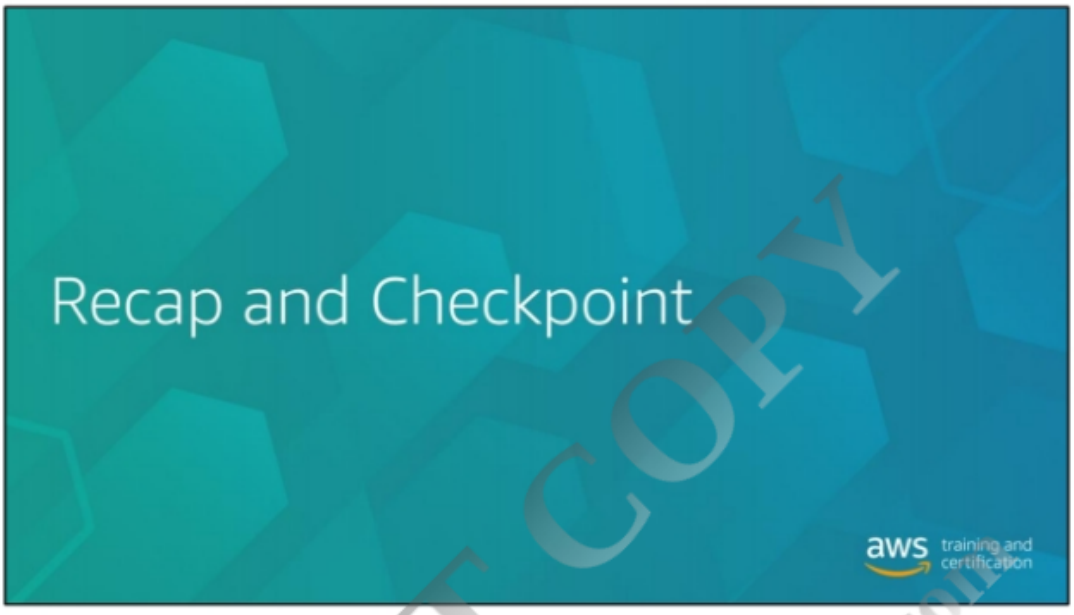
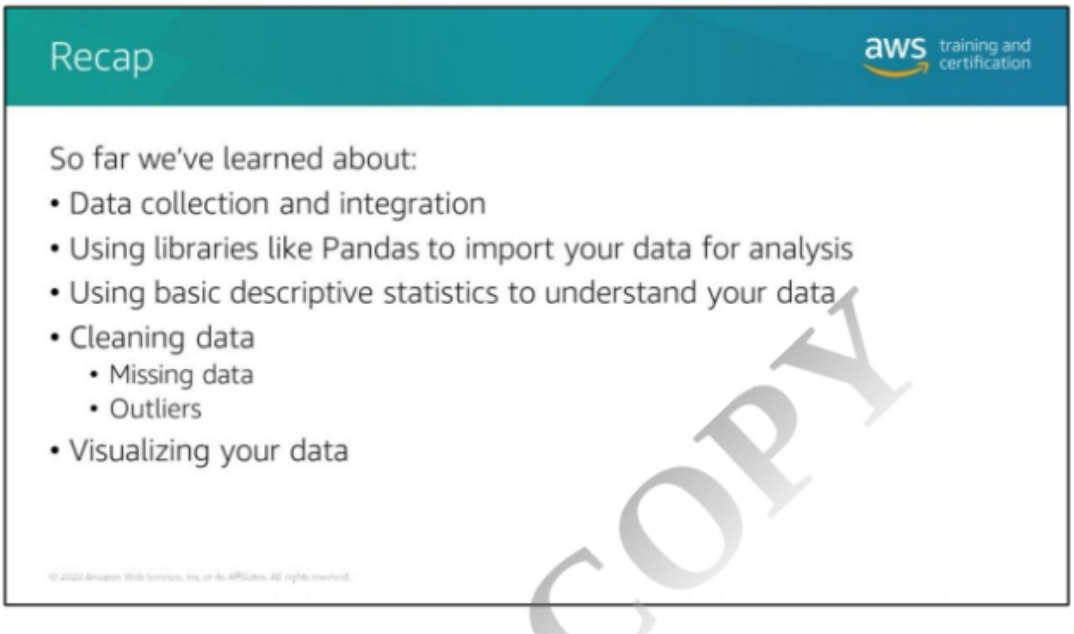




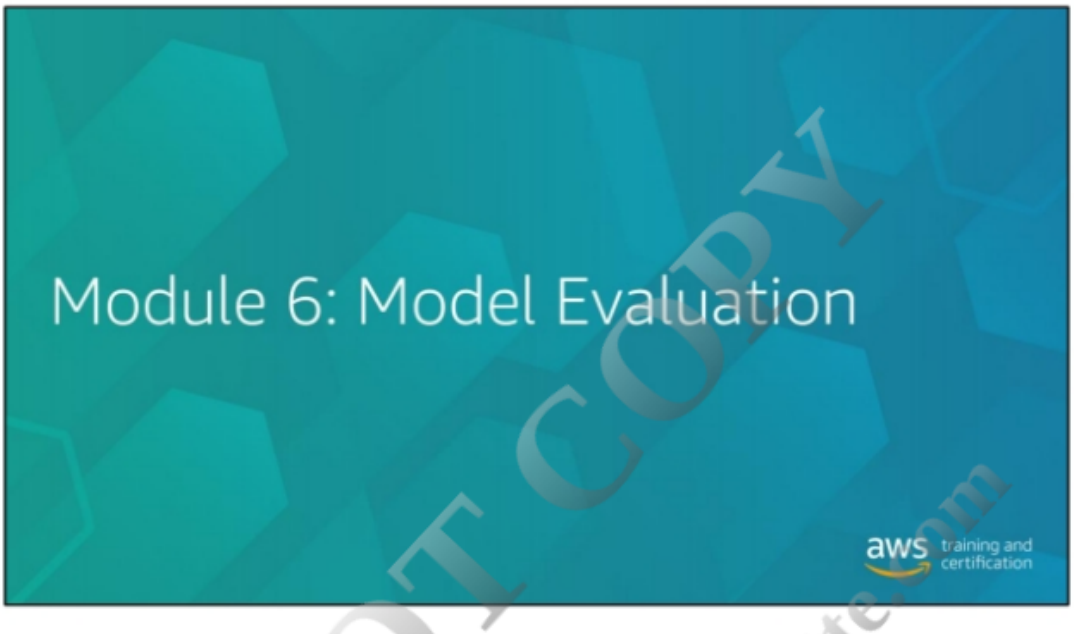


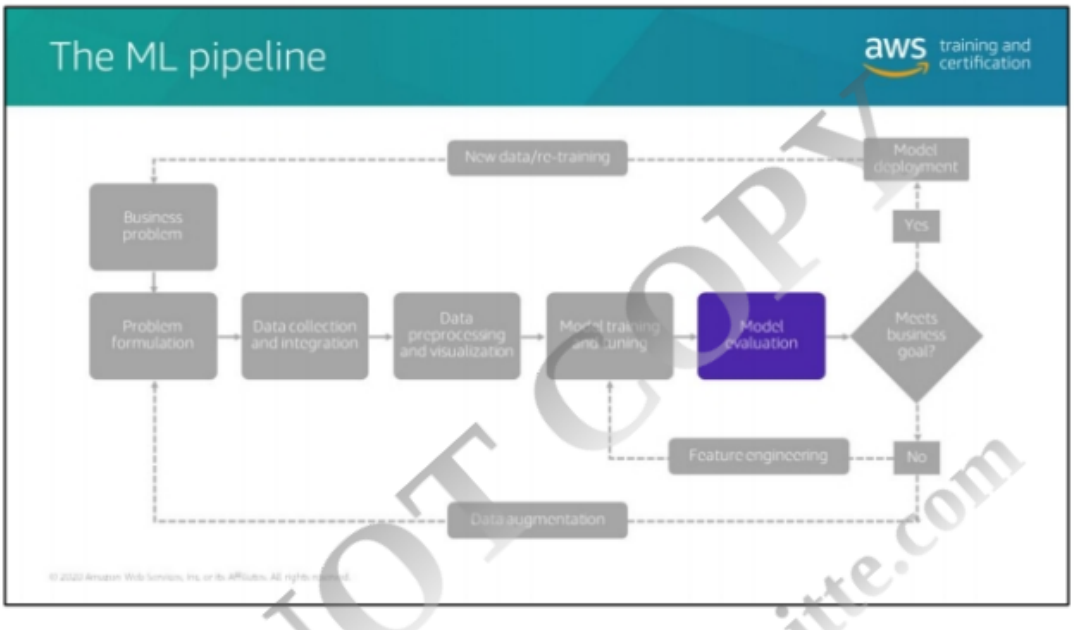

 


https://amazonmr.au1.qualtrics.com/jfe/form/SV_9sn5x7BhaimfbJH



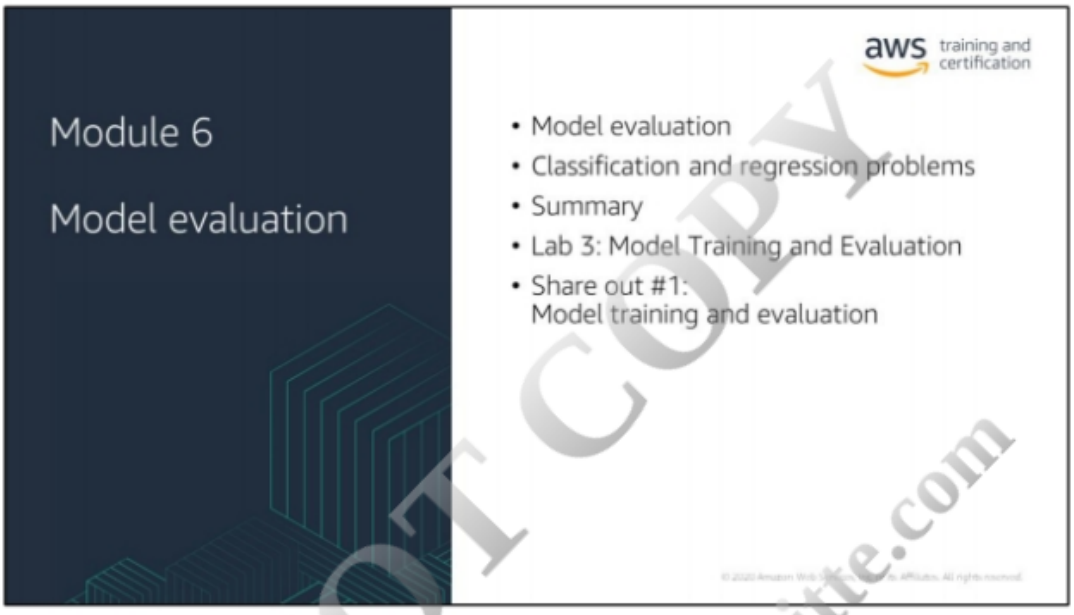





After your model is trained it's time to evaluate it.



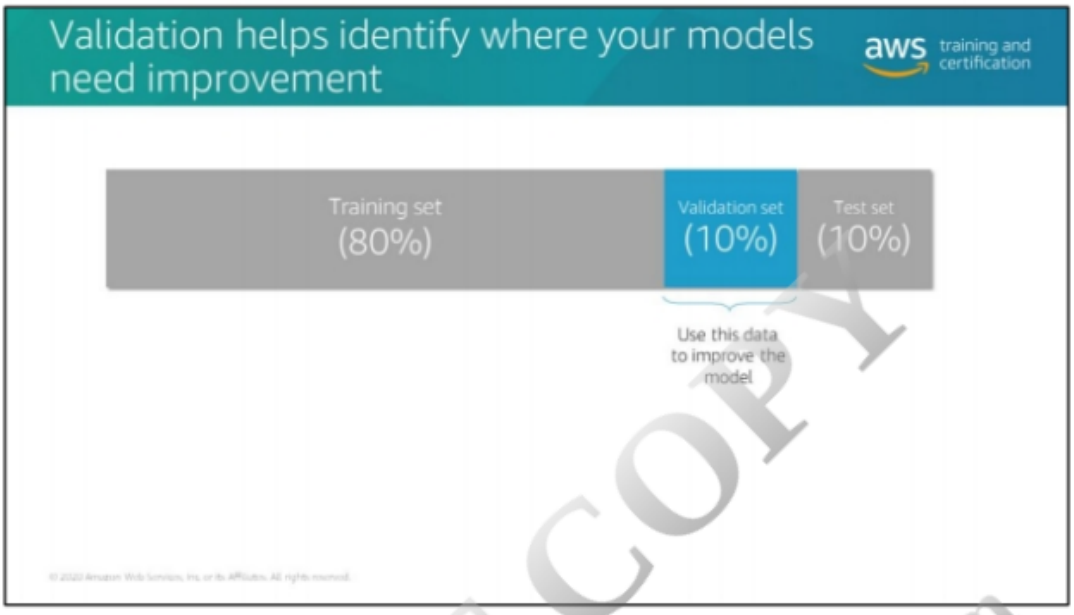
 




This module covers how to evaluate different types of ML problems including classification and regression problems. At the end of the module you'll have time for hands-on practice training and evaluating your model before having an opportunity to present your findings. Don't worry, presenting your findings is 100% optional.



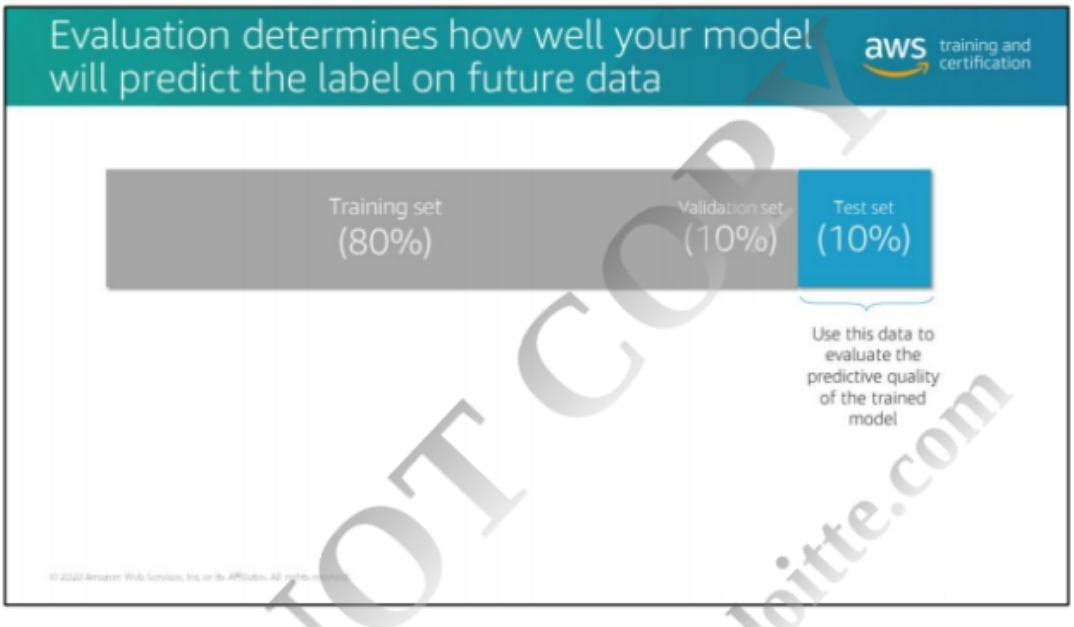
 




In the previous module we trained our models on our training data set. To begin evaluating how your model responds in a non-training environment, you want to start by looking at the data you set aside as your validation set – this is because you want to make sure that the model generalizes to data it has not seen, and because you'll still want to make improvements to the model before determining that it's ready for production.



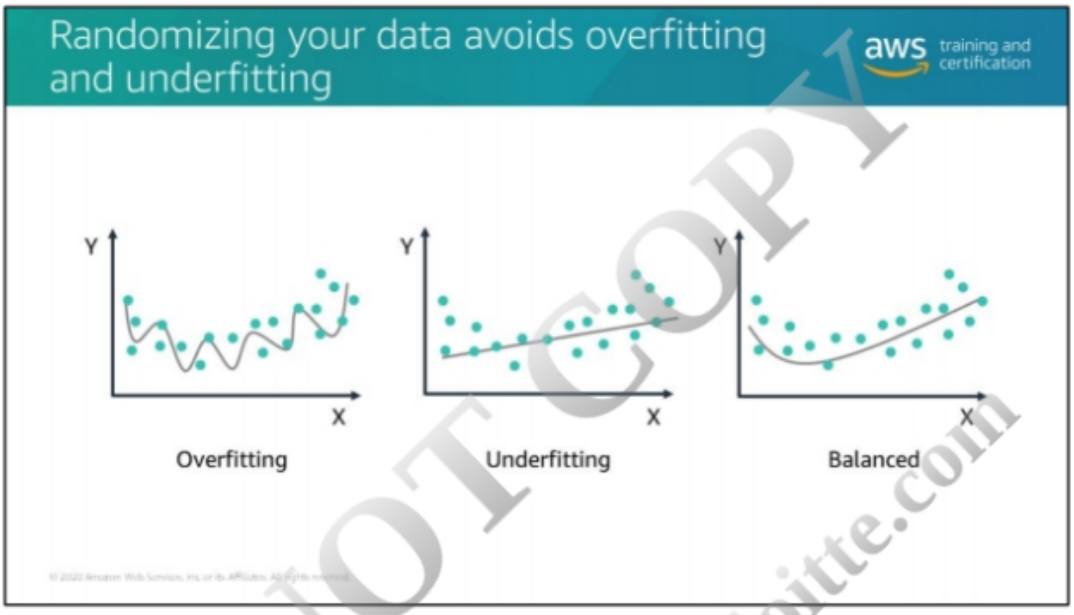
 




Once you've improved your model using that validation data, you're ready to test it one last time to ensure its predictive quality meets your standards. This is why you hold out a sample of your data for validating and evaluating your model.






The bulk of this module will look at the metrics you can use to evaluate your models. However, before doing that, let's revisit the topic of overfitting and underfitting and then introduce an important concept known as the bias/variance tradeoff.

Understanding "model fit" is important for understanding the root cause for poor model accuracy. This understanding will guide you to take corrective steps. We can determine whether a predictive model is underfitting or overfitting the training data by looking at the prediction error on the training data and the evaluation data.

Your model is *underfitting* the training data when the model performs poorly on the training data. This is because the model is unable to capture the relationship between the input examples (often called X) and the target values (often called Y). Your model is *overfitting* your training data when you see that the model performs well on the training data but does not perform well on the evaluation data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

Poor performance on the training data could be because the model is too simple (the input features are not expressive enough) to describe the target well. Performance can be improved by increasing model flexibility. To increase model flexibility, try the following:

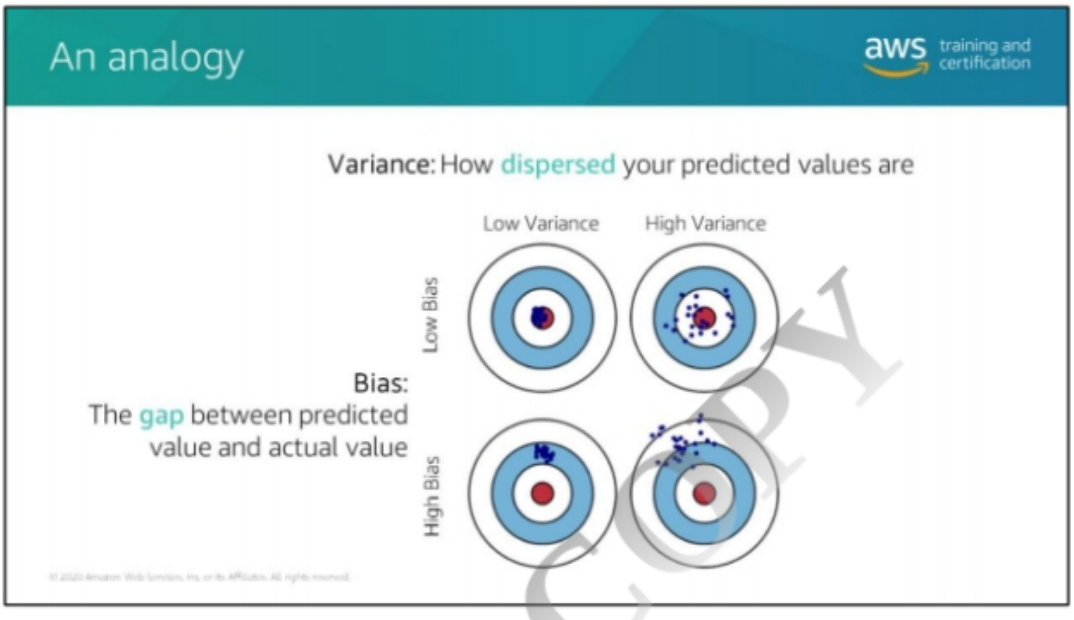
- Add new domain-specific features and more feature Cartesian products, and change the types of feature processing used (e.g., increasing n-grams size)
- Decrease the amount of regularization used

If your model is overfitting the training data, it makes sense to take actions that reduce model flexibility. To reduce model flexibility, try the following:

- Feature selection: consider using fewer feature combinations, decrease n-grams size, and decrease the number of numeric attribute bins.
- Increase the amount of regularization used.

Accuracy on training and test data could be poor because the learning algorithm did not have enough data to learn from.
You could improve performance by doing the following:

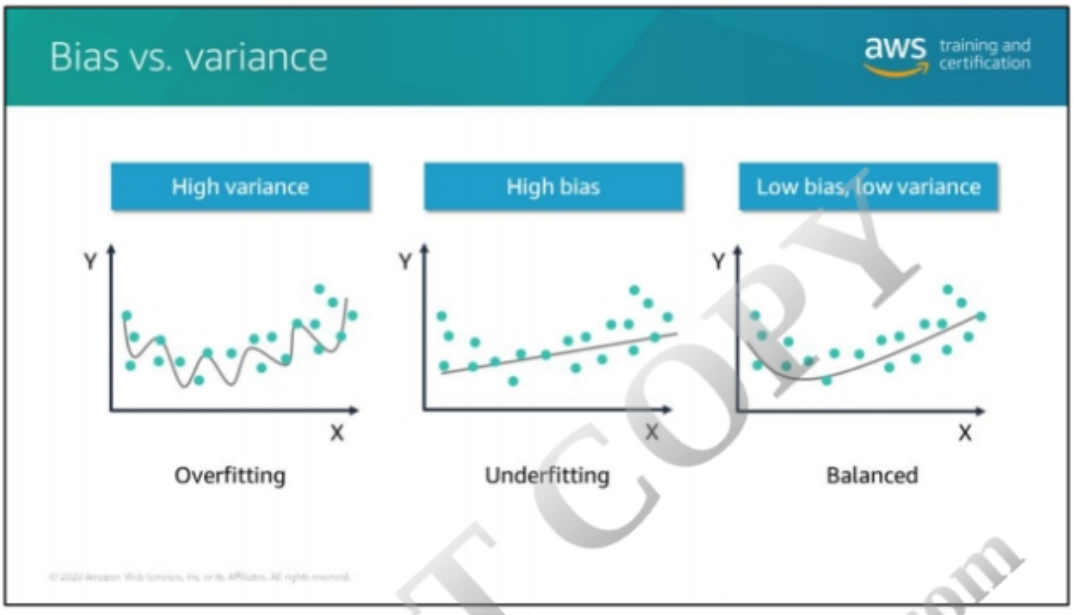
- Increase the amount of training data examples.
- Increase the number of passes on the existing training data.




A bullseye is a nice analogy to use here because generally speaking, the center of the bullseye is where you aim your darts. The center of the bullseye in our situation is the label or target – it predicts the value of your model – and each dot is a result that your model produced during training. So you can see where you've got low bias/high variance (top right image), high bias/high variance (bottom right), and high bias/low variance (bottom left). Ideally – you guessed it – you want low bias and low variance (top left). Realistically though, there's a balancing act happening here. Bias and variance both contribute to errors but what you're ultimately going for is a minimized prediction error – not bias or variance specifically. This is the bias-variance tradeoff.

Bringing underfitting and overfitting back into the picture, *underfitting* is where you have low variance and high bias. These models are overly simple and they can't really see the underlying patterns within the data. *Overfitting* is high variance and low bias (top right). These models are overly complex, and while they can detect patterns in training data, they aren't accurate outside that training data.

As we begin to look more specifically at metrics to use to evaluate your model, keep the bias/variance tradeoff in mind.

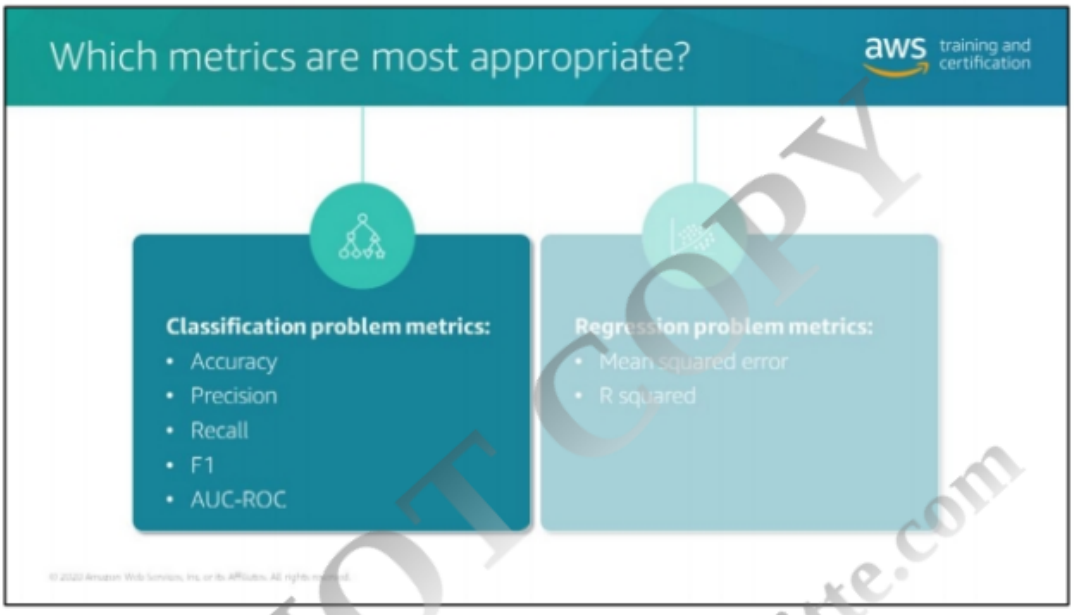



Think about *bias* as the gap between your predicted value and the actual value, whereas *variance* describes how dispersed your predicted values are.

In ML, the ideal algorithm has low bias and can accurately model the true relationship, and it has low variability, by producing consistent predictions across different datasets.



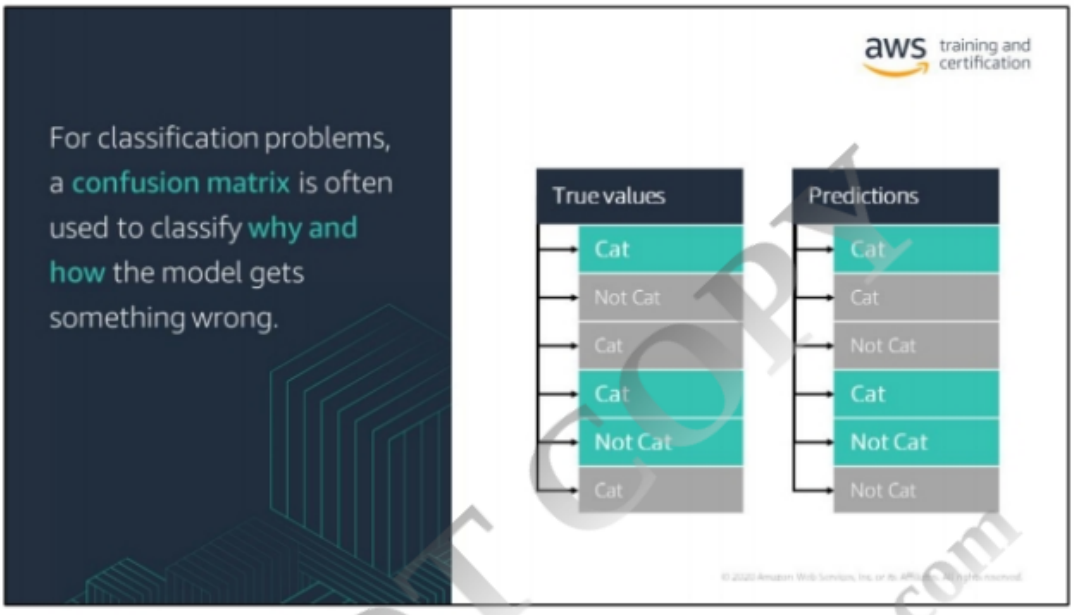
 




In addition to considering your business problem and success metric, the type of ML problem you're working with will influence the model metric you choose. Throughout the rest of this module, we'll look at examples of common metrics used in classification problems, as well as common metrics used in regression problems. Let's start by looking at a simple binary classification problem,



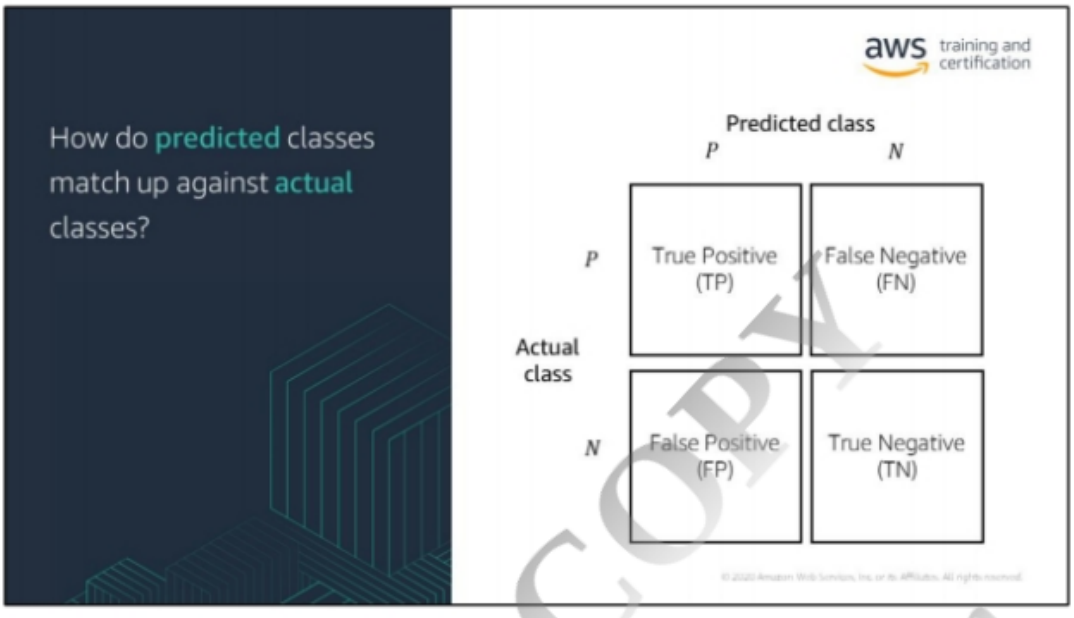
 




In general, evaluating a classification problem like this one works like this: you send the model your held-out observations for which you know the target values. You then compare the predictions returned by the model against the known target value. Finally, you compute a summary metric that tells you how well the predicted and true values match.
A confusion matrix is the building block for running these types of model evaluations for classification problems.



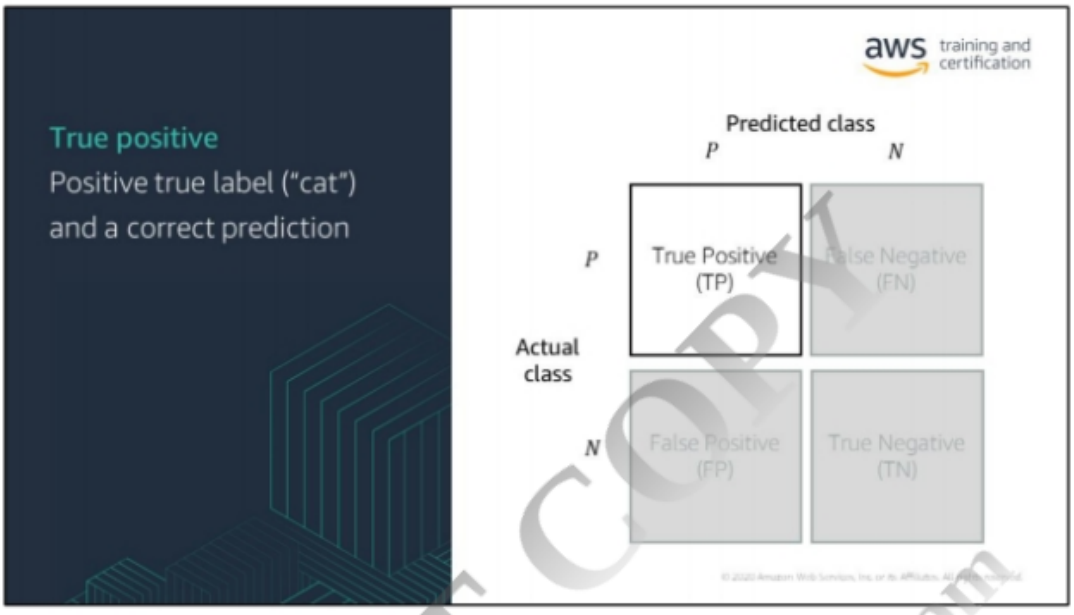
 




Let's dive into a specific example. This is a simple image recognition model that is labeling data as either "cat" or "not cat." In a confusion matrix, you can get a high-level comparison of how the predicted classes matched up against the actual classes. The actual classes run along the left of the confusion matrix. The predicted classes run along the top.



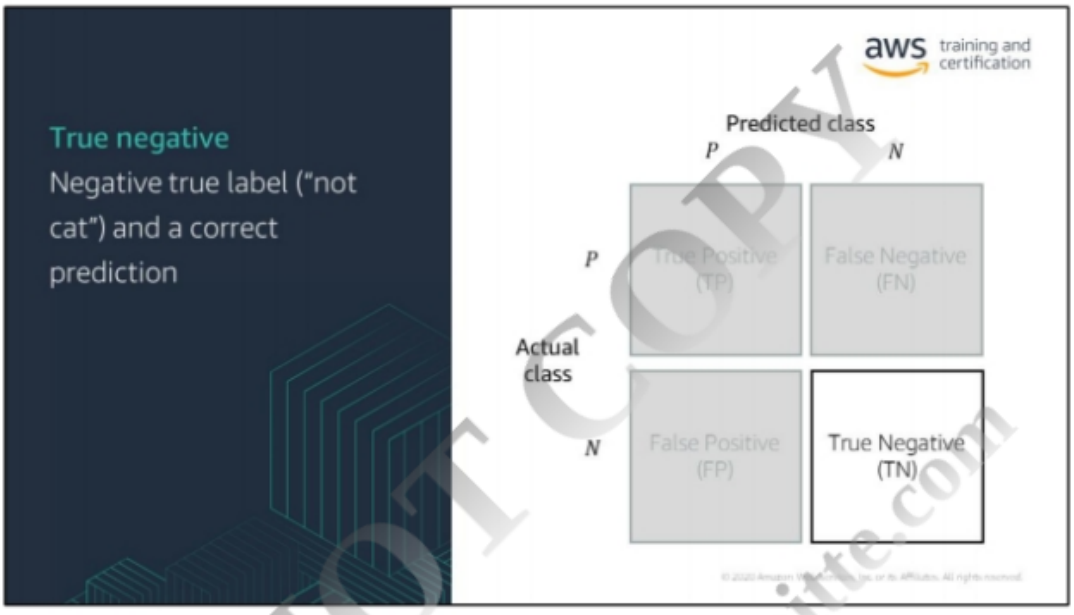
 



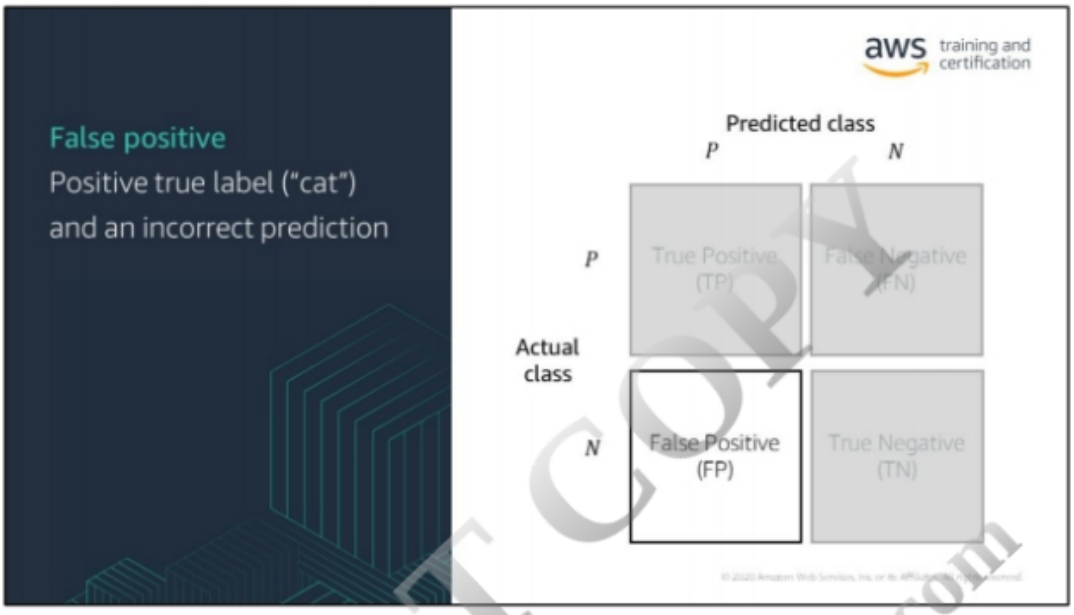

If the actual label or class is "cat," which is identified as "P" for positive in the confusion matrix, and the predicted label or class is also "cat," then you have a True Positive result, which is a good outcome for your model.





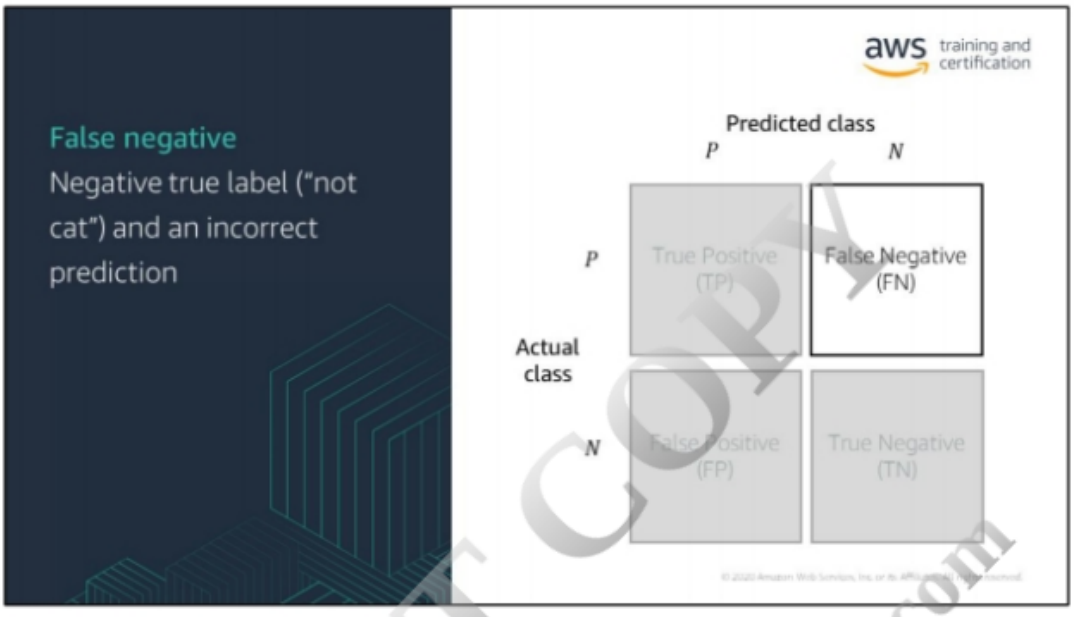

Similarly, if you have an actual label of "not cat," which is identified as N for negative in the confusion matrix, and the predicted label or class is also "not cat," then you have a True Negative, which is also a good outcome for your model. In both of these cases, your model, in using the testing data, predicted the correct outcome.




There are two other possible outcomes, both of which are less than ideal. The first is when the actual class is negative, so "not cat," but the predicted class is positive, so "cat." This is called a False Positive because the prediction is positive, but incorrect.



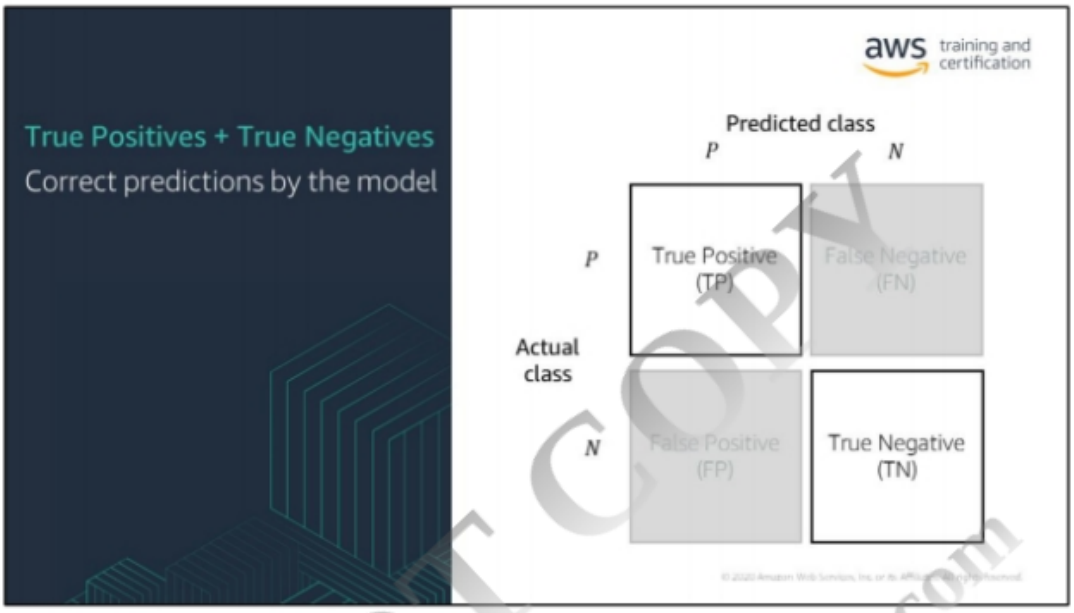
 




A False Negative occurs when the actual class is positive, so "cat," but the predicted class is negative, so "not cat."



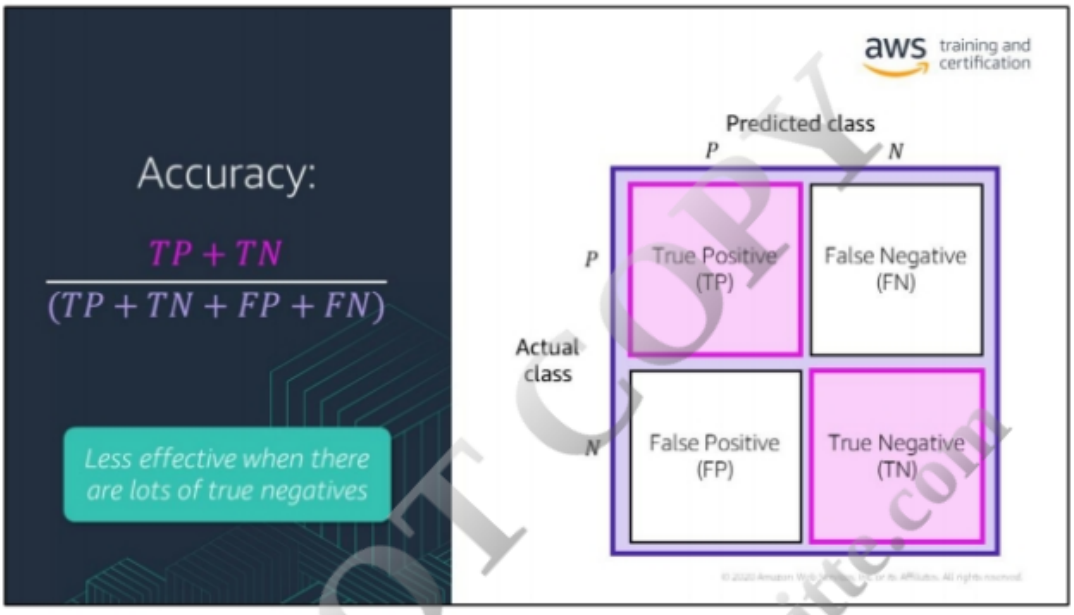
 




Once you apply your model to the testing data, each of these four boxes will include an aggregate number of the unique occurrences of True Positives, False Positives, False Negatives, and True Negatives.
With those four numbers, you can calculate the model's accuracy, also known as its *score*. You can do this by adding up the correct predictions and then dividing that number by the total number of predictions.



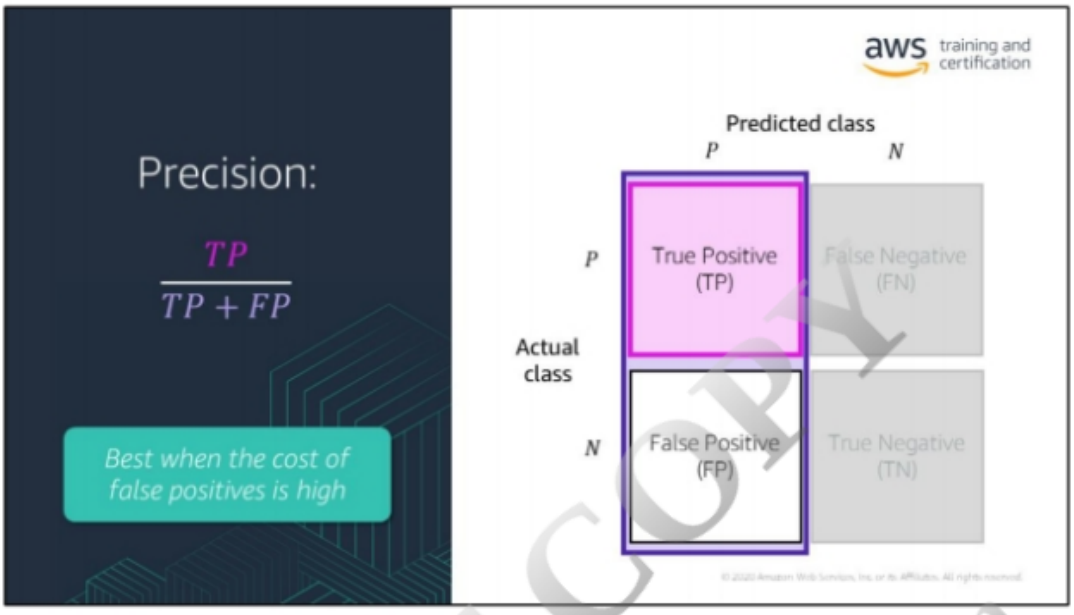



While accuracy is a widely used metric for classification problems, it has limitations. This metric is less effective when there are a lot of true negative cases in your dataset. Think about the "cat/not cat" example. If the vast majority of your accuracy is based on true negatives, all it says about your model is that it is good at predicting what isn't a cat. In this case, you can't feel very confident in your model's ability to predict cats once you roll it out into production.

This leads to an example of why it's so important to ensure that the metric you choose for model evaluation aligns to your business goal. Think about the credit card fraud example we've been discussing throughout this course. Using accuracy as your main metric is probably not a great idea in this case when you have a lot of true negatives. Your really high true negative number might hide the fact that your model's ability to identify cases of fraud (that is, true positives) is less than ideal. As a credit card company, it's probably unacceptable to have less than almost-perfect performance identifying fraud cases, because that would drive customers away, which would be the opposite of what we want to achieve from a business standpoint.



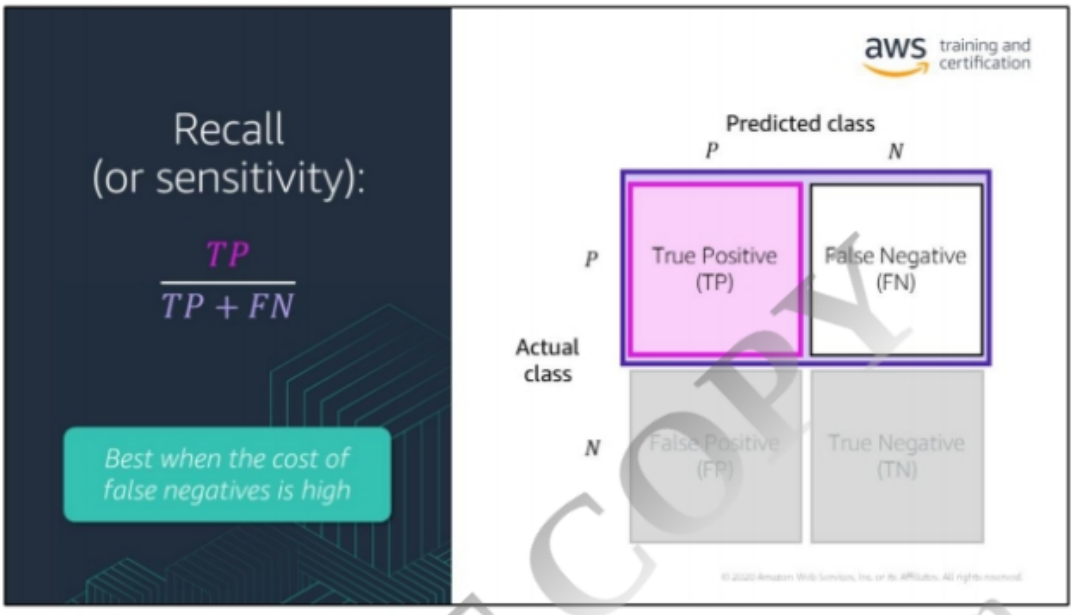



This is why two other metrics are often used in these situations. The first one is *precision*, which essentially just removes the negative predictions from the picture. Precision is the proportion of positive predictions that are actually correct. You can calculate it by taking the true positive and dividing it by true positive plus false positive.

When the cost of false positives are high in your particular business situation, precision may be a good metric. Think about a classification model that identifies emails as spam or not. In this case, you do not want your model labeling an email as spam and therefore preventing your users from seeing that email, when in reality the email was legitimate.



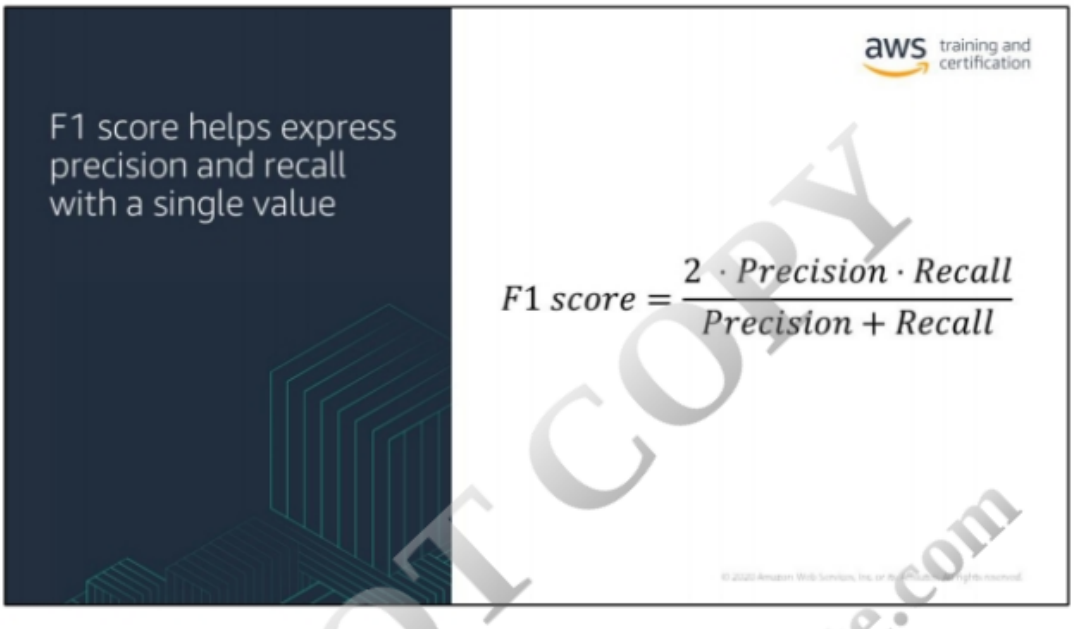



In addition to precision, there is also *recall*. In recall, you are looking at the proportion of correct sets that are identified as positive. Recall is calculated by dividing true positive by [true positive plus false negative]. By looking at that ratio, you get an idea of how good the algorithm is at detecting, for example, cats.

Think about an example of a model that needs to predict whether or not a patient has a terminal illness or not. In this case, using precision as your evaluation metric, does not account for the false negatives in your model. In this situation, it is extremely important and vital to the success of the model to not identify the illness in a patient who, in reality, actually has that illness. Recall in this situation is a better metric to use.



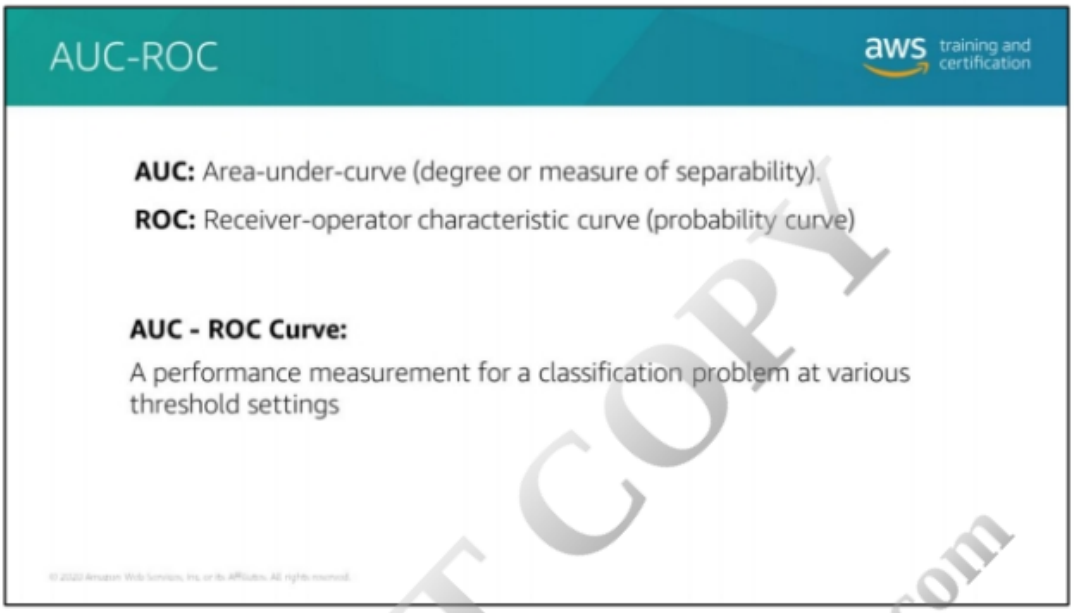
 




But it doesn't always have to be one or the other. The F1 Score combines precision and recall together to give you just one number to quantify the overall performance of a particular ML algorithm. You should look at using F1 score when you have a class imbalance but want to preserve the equality between precision and recall



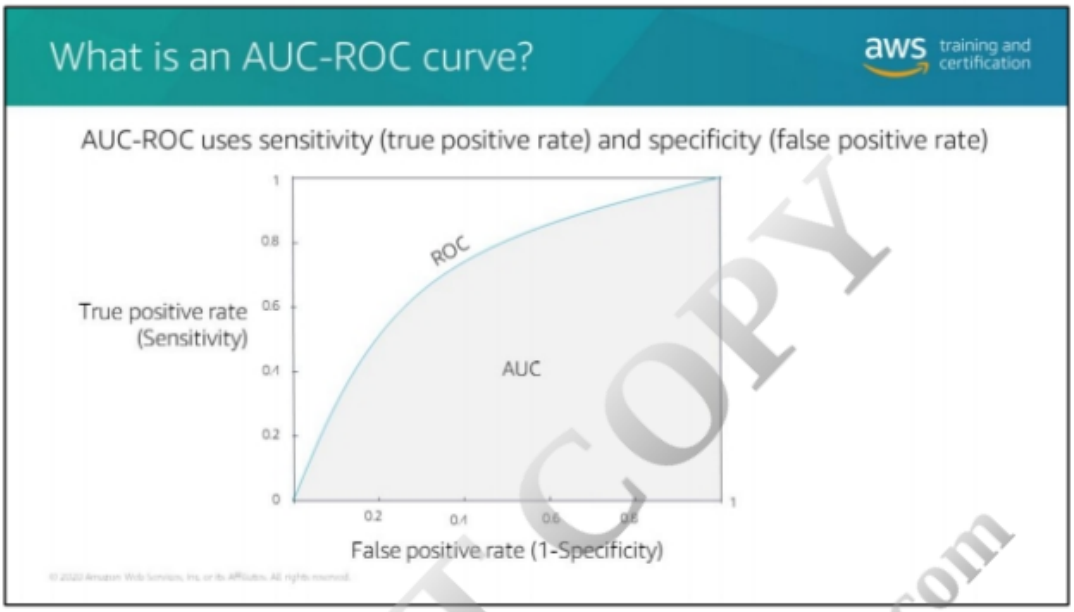
 




Area under the curve-Receiver operator curve (AUC-ROC) is another evaluation metric. ROC is a probability curve and AUC represents the degree or measure of separability. In general, AUC-ROC curve can show what the curve for true positive vs false positive looks like at various thresholds. That means that when you calculate the AUC-ROC curve, you plot multiple confusion matrices at different thresholds and compare them to one another to find out the threshold you need for your business use case.



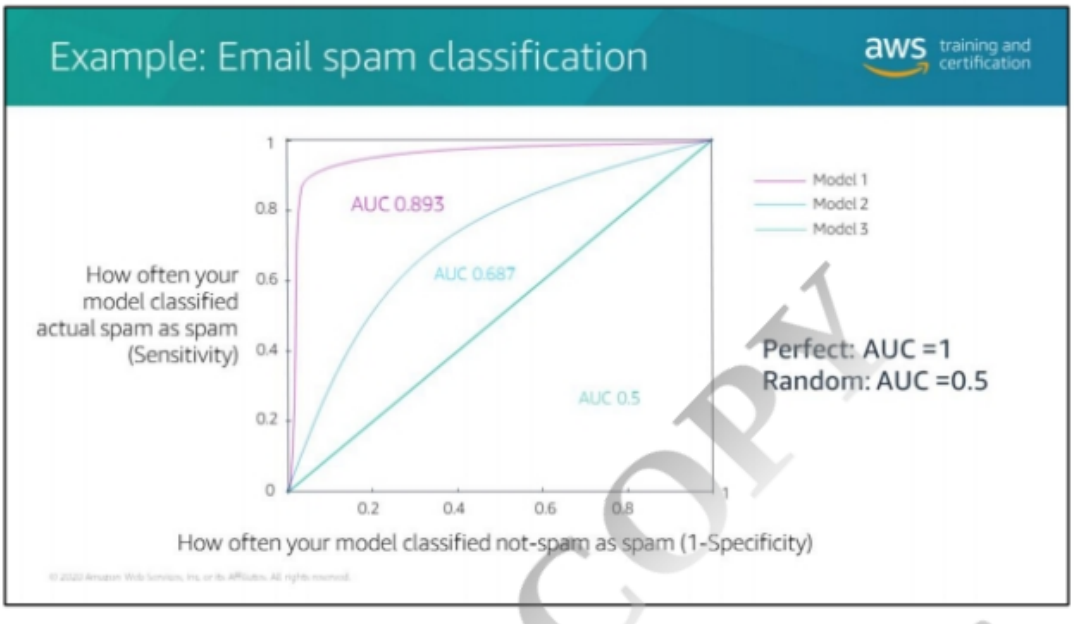
 




The AUC-ROC curve is the area under the Receiver operator characteristic curve. The ROC curve plots the binary classifier at different thresholds for the binary classifier. ROC-AUC curve helps in selecting good classifiers. Lets take a look at that in an example.





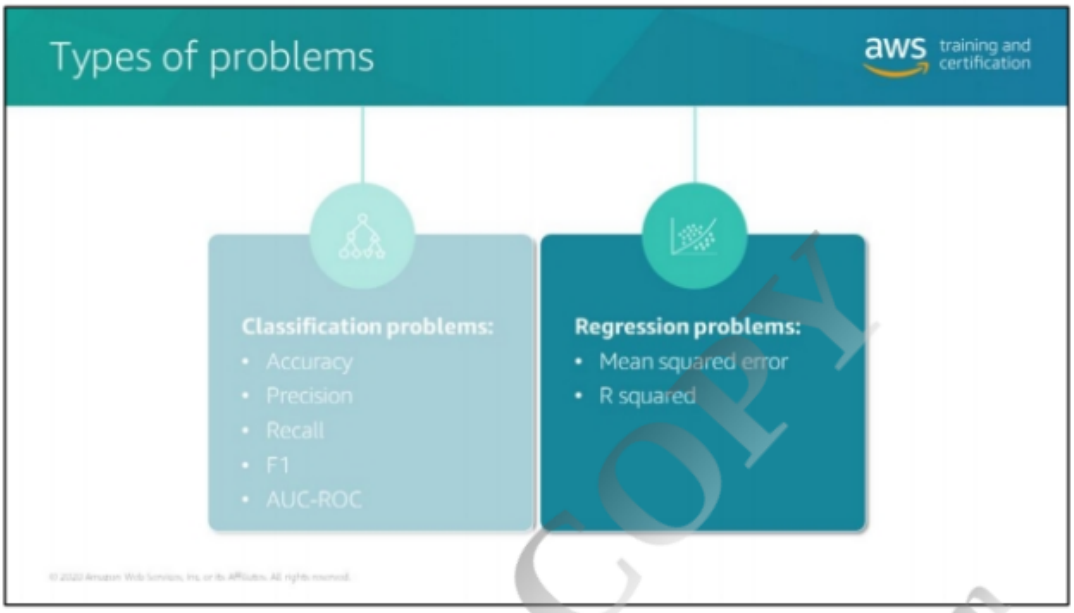

Take an example of email spam classification. The emails are rank ordered by the classifier's risk score. On the left is high scoring emails, on the right is the vast majority of low scoring emails. And here is the tradeoff curve. On the X-axis is the percentage of good emails that are going to be affected by our action – in this case, sidelining emails into spam folder. On the y-axis is the percentage of spam we are capturing. That knee in the curve is a great tradeoff point.

If we set our operating cutoff at that point, we'll get a good balance between the good population we impact, and the bads we capture. Of course, if there are different costs to sidelining a population and capturing the bads, we can shift that operating point to the left or to the right. Good classifiers produce good tradeoff curves.

As you improve the features you use as inputs to the classifier, or as you improve the algorithm, the curve shifts up and to the left. The perfect tradeoff curve of course is one that goes along the upper left of the rectangle






But what if you're dealing with a regression problem? In that case, there are other common metrics you can use to evaluate your model, including: mean squared error and R squared.



## Mean Squared Error (MSE)

aws training and certification

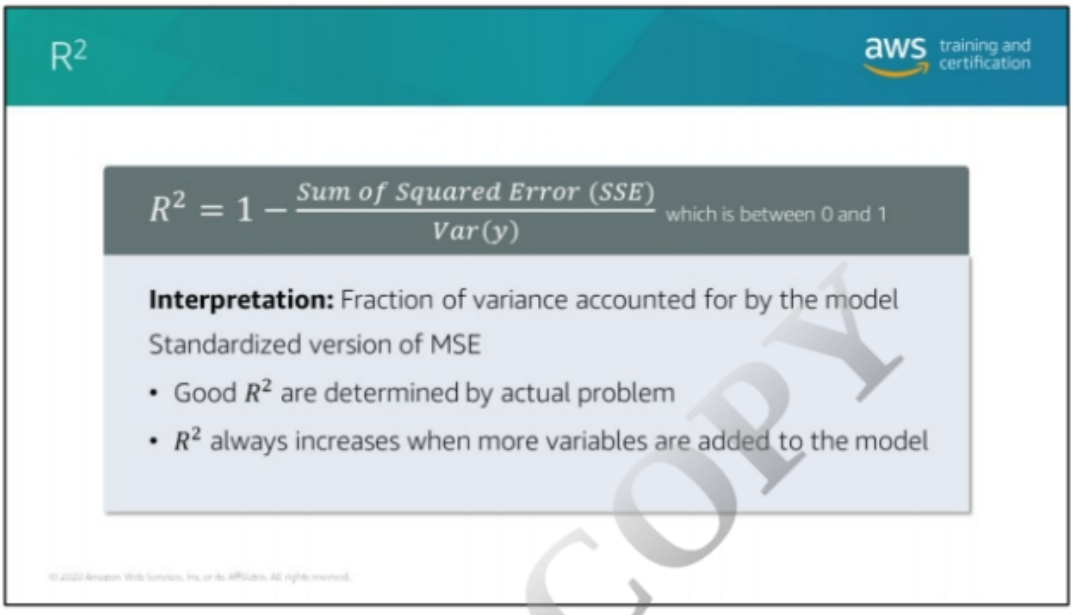
$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$


Mean squared error is very commonly used. Its general purpose is the same as what we saw with classification metrics. We determine the prediction from the model and we compare the difference between the prediction and the actual outcome.

More specifically, we take the difference between the prediction and actual value, square that difference, and then sum up all the squared differences for all the observations. In scikit-learn, you can use the mean squared error function directly from the metrics library.



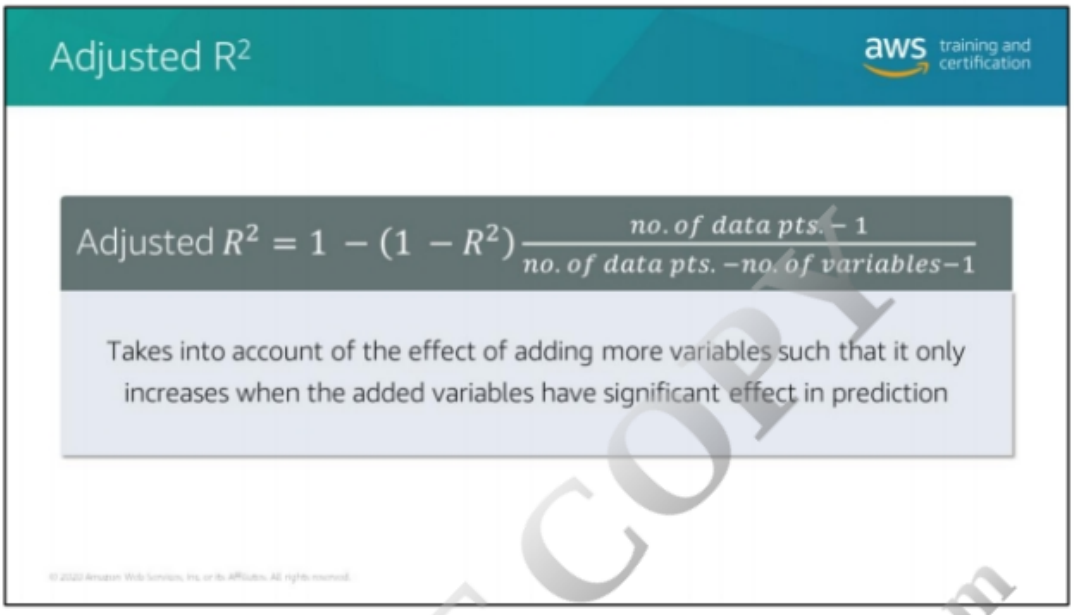
 




R squared is another commonly used metric with linear regression problems. R squared explains the fraction of variance accounted for by the model. It's like a percentage, reporting a number from 0 to 1. When R squared is close to 1 it usually indicates that a lot of the variabilities in the data can be explained by the model itself.



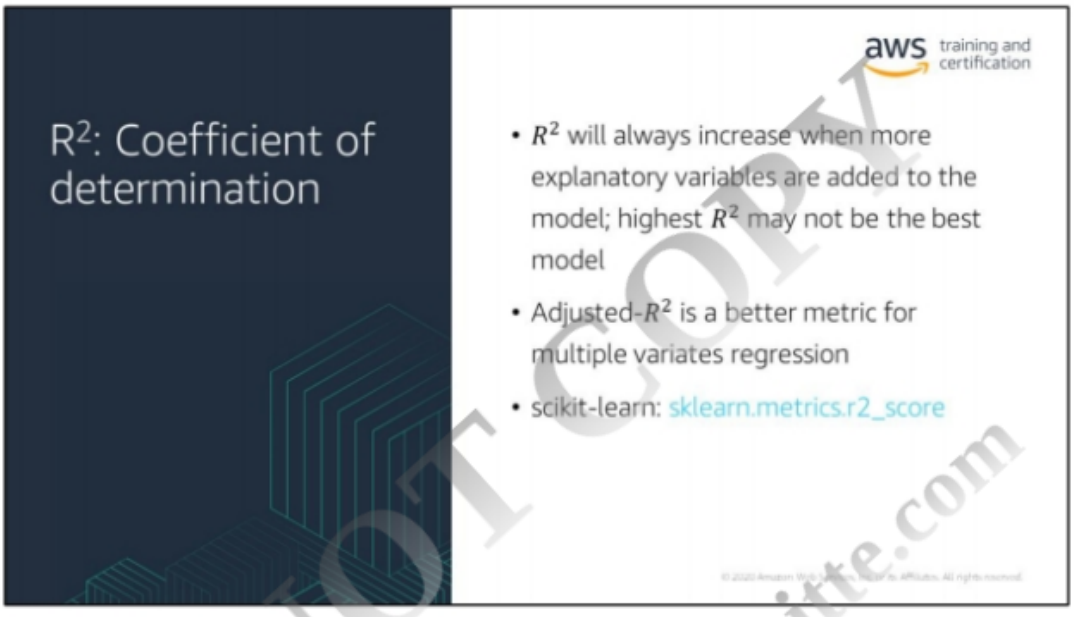



Depending on your business problem, a good R squared result could be 0.6 (60%) or 0.7 (70%). For other business problems, a good R squared might be 0.8 (80%) or 0.85 (85%). So, the threshold for a good R squared really depends on your business problem. Some business problems are very difficult to achieve a high R squared.

Relatedly, R squared is always increasing when more variables are added to the model, which sometimes leads to overfitting. It isn't always that the higher the R squared, the better the model. We have to balance the overfitting problem.



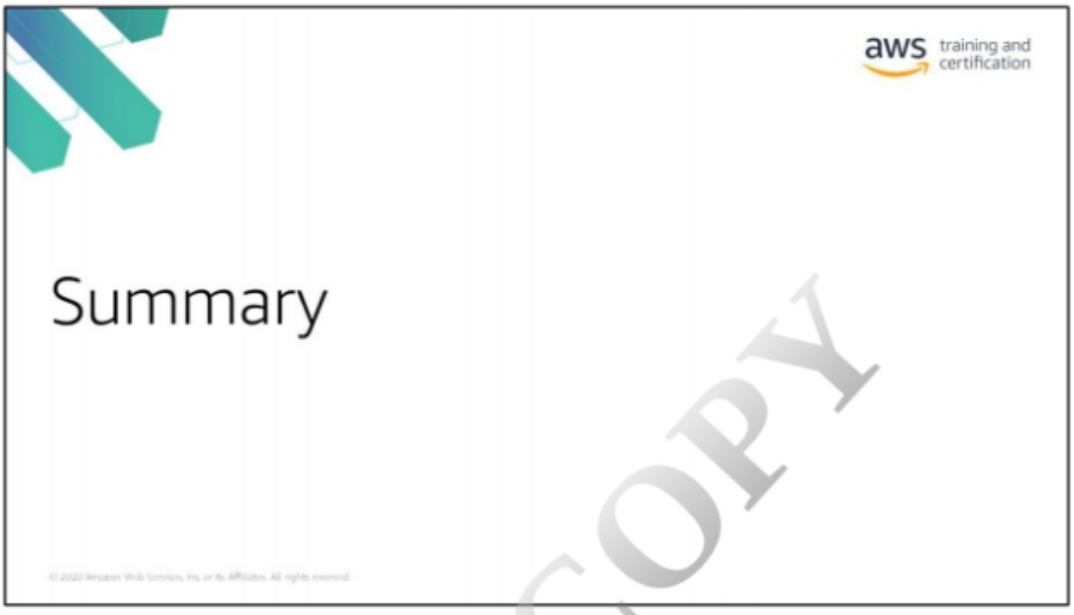


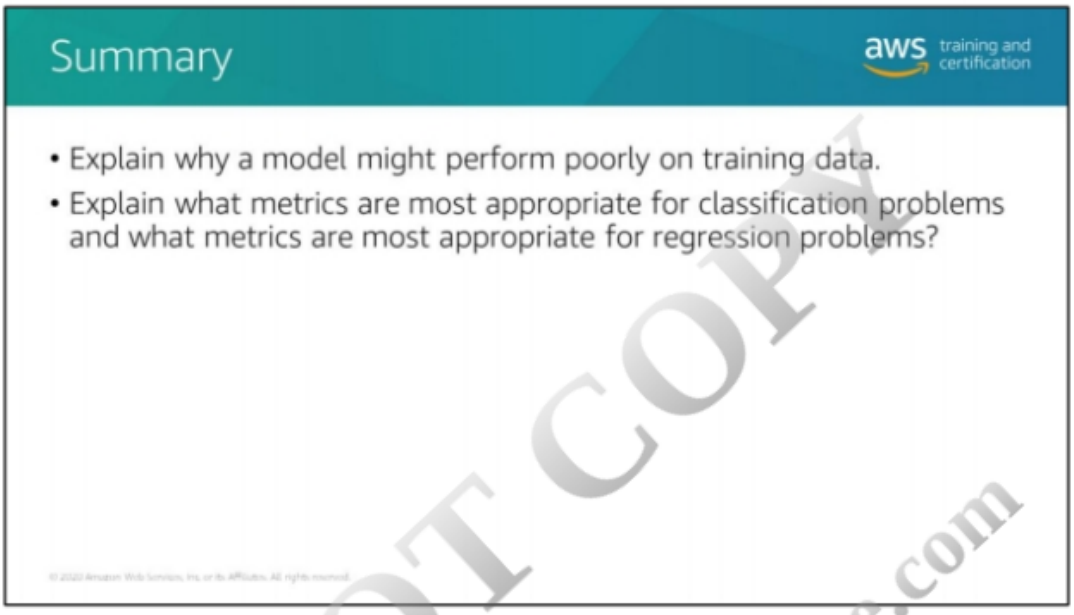

To counter this potential issue, there is another metric called the Adjusted R squared. The Adjusted R squared has already taken care of the added effect for additional variables and it only increases when the added variables have significant effects in the prediction. The adjusted R squared adjusts your final value based on the number of features and number of data points you have in your dataset.

A recommendation, therefore, is to look at both R squared and Adjusted R squared. This will ensure that your model is performing well but that there's also not too much overfitting.








Explain why a model might perform poorly on training data.

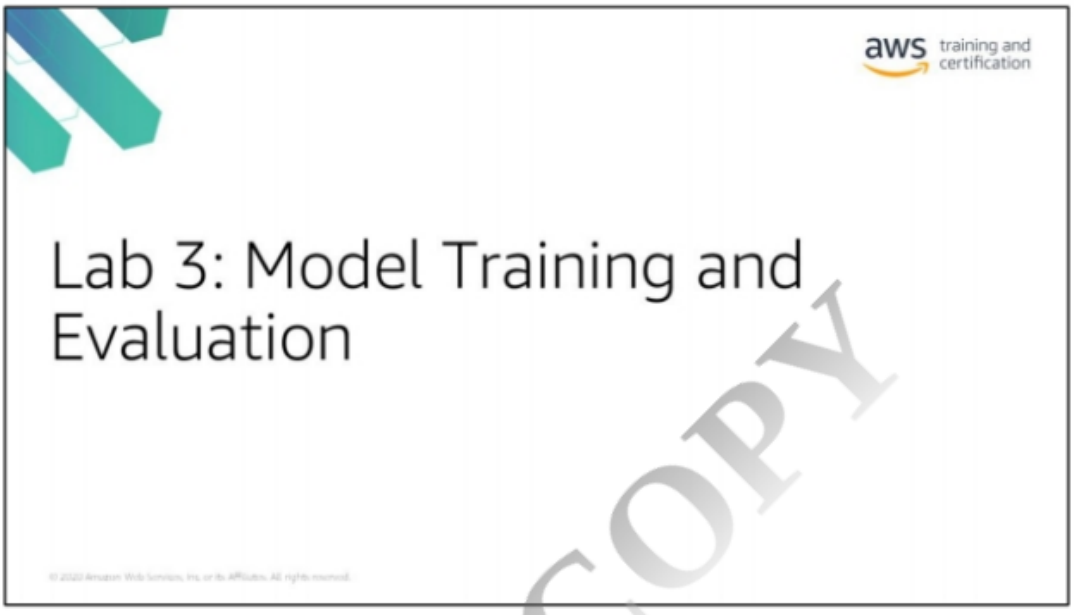
- When the model performs poorly on training data it is likely the result of underfitting. This is because the model is unable to capture the relationship between the input examples (often called X) and the target values (often called Y).

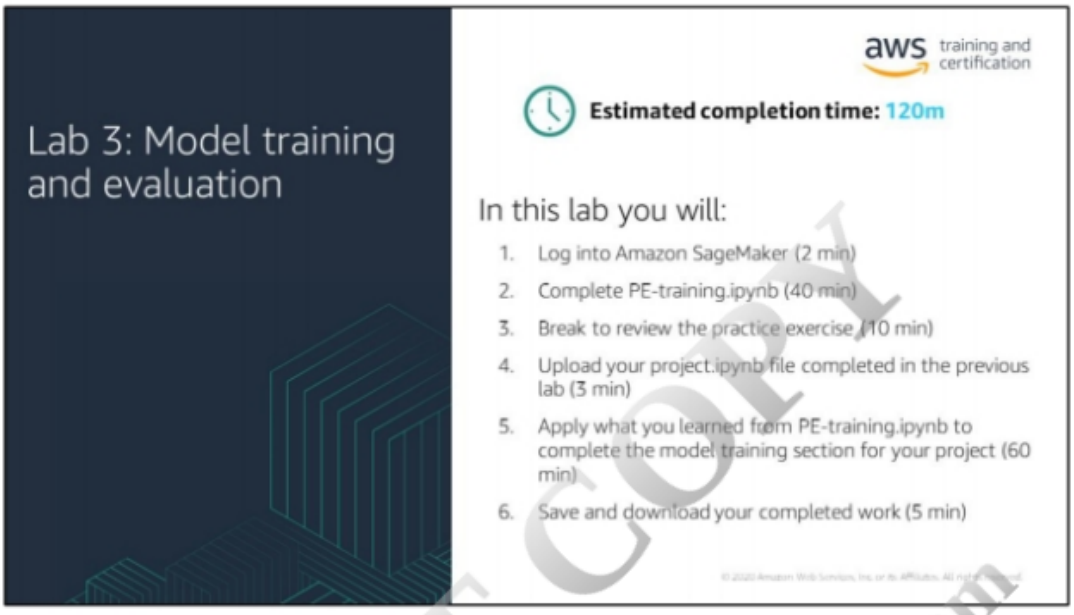
Explain what metrics are most appropriate for classification problems and what metrics are most appropriate for regression problems?

- Classification problem metrics include Accuracy: Precision, Recall, F1, AUC-ROC
- Regression problem metrics include Mean squared error and R squared





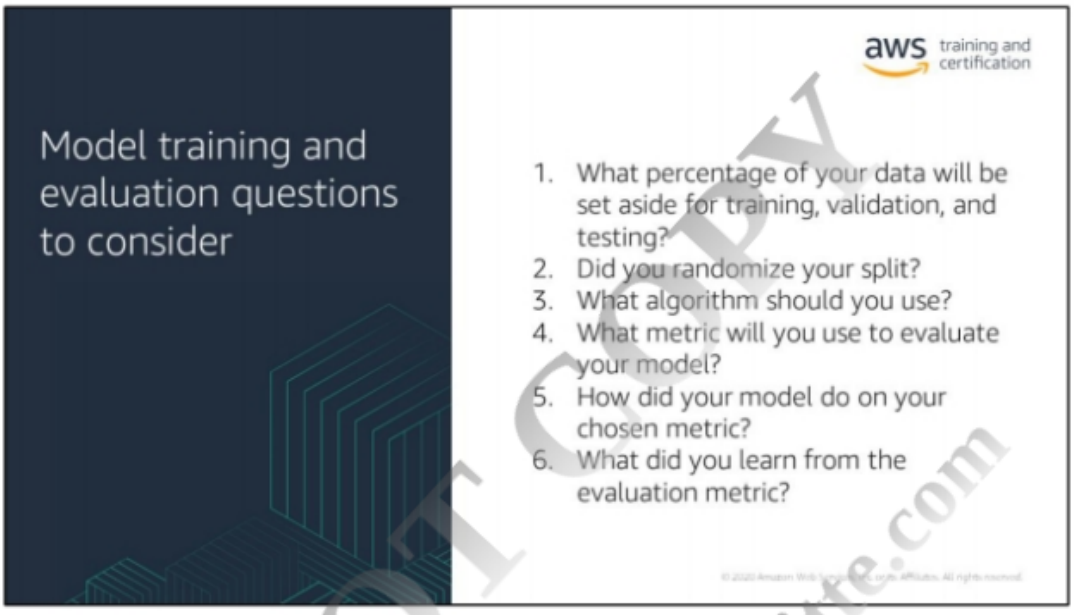



To train your model using Amazon SageMaker training jobs, let's get back into the SageMaker console. There we'll configure our job, select an algorithm, set a few hyperparameters, dive into our datasets, and then create the training job. After that, you'll deploy it to an endpoint and then evaluate it based on identified success metrics.

Note: Be sure to take the break to review the practice exercise.






As you work through model training and evaluation for the business scenario you chose, be sure to track any relevant findings in your project template.